Mellanox ScalableSHMEM User Manual

Rev 2.2

www.mellanox.com

NOTE:
THIS HARDWARE, SOFTWARE OR TEST SUITE PRODUCT (“PRODUCT(S)”) AND ITS RELATED DOCUMENTATION ARE PROVIDED BY MELLANOX TECHNOLOGIES “AS-IS” WITH ALL FAULTS OF ANY KIND AND SOLELY FOR THE PURPOSE OF AIDING THE CUSTOMER IN TESTING APPLICATIONS THAT USE THE PRODUCTS IN DESIGNATED SOLUTIONS. THE CUSTOMER'S MANUFACTURING TEST ENVIRONMENT HAS NOT MET THE STANDARDS SET BY MELLANOX TECHNOLOGIES TO FULLY QUALIFY THE PRODUCTO(S) AND/OR THE SYSTEM USING IT. THEREFORE, MELLANOX TECHNOLOGIES CANNOT AND DOES NOT GUARANTEE OR WARRANT THAT THE PRODUCTS WILL OPERATE WITH THE HIGHEST QUALITY. ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL MELLANOX BE LIABLE TO CUSTOMER OR ANY THIRD PARTIES FOR ANY DIRECT, INDIRECT, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES OF ANY KIND (INCLUDING, BUT NOT LIMITED TO, PAYMENT FOR PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY FROM THE USE OF THE PRODUCT(S) AND RELATED DOCUMENTATION EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Mellanox Technologies
350 Oakmead Parkway Suite 100
Sunnyvale, CA 94085
U.S.A.
www.mellanox.com
Tel: (408) 970-3400
Fax: (408) 970-3403

Mellanox Technologies, Ltd.
Beit Mellanox
PO Box 586 Yokneam 20692
Israel
www.mellanox.com
Tel: +972 (0)74 723 7200
Fax: +972 (0)4 959 3245

© Copyright 2012. Mellanox Technologies. All Rights Reserved.

Mellanox®, Mellanox logo, BridgeX®, ConnectX®, CORE-Direct®, InfiniBridge®, InfiniHost®, InfiniScale®, PhyX®, SwitchX®, Virtual Protocol Interconnect® and Voltaire® are registered trademarks of Mellanox Technologies, Ltd.

Connect-IB™, FabricIT™, MLNX-OS™, MetroX™, ScalableHPC™, Unbreakable-Link™, UFM™ and Unified Fabric Manager™ are trademarks of Mellanox Technologies, Ltd.

All other trademarks are property of their respective owners.

Table of Contents

Table of Contents 3
Document Revision History 4
Chapter 1 Shared Memory Access Overview 5
1.1 Mellanox ScalableSHMEM 5
Chapter 2 Installing ScalableSHMEM. 6
2.1 Installing ScalableSHMEM 6
2.2 Compiling ScalableSHMEM Application 6
2.3 Running ScalableSHMEM Application 6
2.3.1 Basic ScalableSHMEM Job Run Example 7
Chapter 3 Running ScalableSHMEM 8
3.1 Running ScalableSHMEM with MXM 8
3.1.1 Enabling MXM for ScalableSHMEM Jobs 8
3.1.2 Working with Multiple HCAs 8
3.2 Running ScalableSHMEM with RC Transport 9
3.2.1 Working with Multiple HCAs 9
3.3 Running ScalableSHMEM with FCA 9
3.4 Developing Application using ScalableSHMEM together with MPI 10
3.5 ScalableSHMEM Tunable Parameters 10
3.5.1 OpenSHMEM MCA Parameters for Symmetric Heap Allocation 11
3.5.2 Parameters Used to Force Connection Creation 11
3.6 Configuring ScalableSHMEM with External Profiling Tools (TAU) 12
3.6.1 Using TAU with OpenSHMEM 12
Appendix A Performance Optimization 13
A.1 Configuring Hugepages 13
A.2 Tuning MTU Size to the Recommended Value 13
A.3 HPC Applications on Intel Sandy Bridge Machines 14

Document Revision History

Table 1 - Document Revision History

Revision	Date	Change
2.2	December 212	**Removed the following sections:** • ScalableSHMEM Supported Platforms and Operating Systems. Added it to the ScalableSHMEM Release Notes **Updated the following sections:** • "Installing ScalableSHMEM" - updated ScalableSHMEM version and installation path • "Basic ScalableSHMEM Job Run Example" - updated ScalableSHMEM version • "Enabling MXM for ScalableSHMEM Jobs"- added the option of how to force MXM usage • "Running ScalableSHMEM with FCA" on page 9 - updated examples • "ScalableSHMEM Tunable Parameters" on page 10 - updated the list of the tunable parameters **Added the following sections:** • "Working with Multiple HCAs" on page 8 • "Running ScalableSHMEM with RC Transport" on page 9 • "Working with Multiple HCAs" on page 9 • "Developing Application using ScalableSHMEM together with MPI" on page 10 • "OpenSHMEM MCA Parameters for Symmetric Heap Allocation" on page 11 • "Parameters Used to Force Connection Creation" on page 11
2.1	June 2012	Initial release

1 Shared Memory Access Overview

The Shared Memory Access (SHMEM) routines provide low-latency, high-bandwidth communication for use in highly parallel scalable programs. The routines in the SHMEM Application Programming Interface (API) provide a programming model for exchanging data between cooperating parallel processes. The SHMEM API can be used either alone or in combination with MPI routines in the same parallel program.

The SHMEM parallel programming library is an easy-to-use programming model which uses highly efficient one-sided communication APIs to provide an intuitive global-view interface to shared or distributed memory systems. SHMEM's capabilities provide an excellent low level interface for PGAS applications.

A SHMEM program is of a single program, multiple data (SPMD) style. All the SHMEM processes, referred as processing elements (PEs), start simultaneously and run the same program. Commonly, the PEs perform computation on their own sub-domains of the larger problem, and periodically communicate with other PEs to exchange information on which the next communication phase depends.

The SHMEM routines minimize the overhead associated with data transfer requests, maximize bandwidth, and minimize data latency (the period of time that starts when a PE initiates a transfer of data and ends when a PE can use the data).

SHMEM routines support remote data transfer through:

- "put" operations - data transfer to a different PE
- "get" operations - data transfer from a different PE, and remote pointers, allowing direct references to data objects owned by another PE

Additional supported operations are collective broadcast and reduction, barrier synchronization, and atomic memory operations. An atomic memory operation is an atomic read-and-update operation, such as a fetch-and-increment, on a remote or local data object.

SHMEM libraries implement active messaging. The sending of data involves only one CPU where the source processor puts the data into the memory of the destination processor. Likewise, a processor can read data from another processor's memory without interrupting the remote CPU. The remote processor is unaware that its memory has been read or written unless the programmer implements a mechanism to accomplish this.

1.1 Mellanox ScalableSHMEM

The ScalableSHMEM programming library is a one-side communications library that supports a unique set of parallel programming features including point-to-point and collective routines, synchronizations, atomic operations, and a shared memory paradigm used between the processes of a parallel programming application.

Mellanox ScalableSHMEM is based on the API defined by the OpenSHMEM.org consortium. The library works with the OpenFabrics RDMA for Linux stack (OFED), and also has the ability to utilize MellanoX Messaging libraries (MXM) as well as Mellanox Fabric Collective Accelerations (FCA), providing an unprecedented level of scalability for SHMEM programs running over InfiniBand.

The latest ScalableSHMEM software can be downloaded from the Mellanox website.

2 Installing ScalableSHMEM

2.1 Installing ScalableSHMEM

MLNX_OFED v1.8 includes ScalableSHMEM 2.2 which is installed under: `/opt/mellanox/openshmem/2.2.`
If you have installed OFED 1.8, you do *not* need to download and install ScalableSHMEM.

The ScalableSHMEM package is an .rpm file and should be installed on all cluster nodes. It is built to support the SLURM job scheduler by utilizing a PMI API.

➢ ***To install ScalableSHMEM, perform the following steps:***

1. Login as root. Run:

```
# rpm -ihv openshmem-2.2-XXXXX.x86_64.rpm --nodeps
```

2.2 Compiling ScalableSHMEM Application

The ScalableSHMEM package contains a `shmemcc` utility which is used as a compiler command.

➢ ***To compile ScalableSHMEM application:***

1. Save the code example below as a file called `"example.c"`.

```
#include <stdio.h>
#include <stdlib.h>
#include <shmem.h>
int main(int argc, char **argv)
{
    int my_pe, num_pe;
    shmem_init();
    my_pe = my_pe();
    num_pe = num_pes();
    printf("Hello World from process %d of %d\n", my_pe, num_pes);
    exit(0);
}
```

2. Compile the example with the SHMEM C wrapper compiler.

```
% /opt/mellanox/openshmem/2.2/bin/shmemcc -o example.exe example.c
```

2.3 Running ScalableSHMEM Application

The ScalableSHMEM framework contains the `shmemrun` utility which launches the executable from a service node to compute nodes. This utility accepts the same command line parameters as `mpirun` from the OpenMPI package.

For further information, please refer to OpenMPI MCA parameters documentation at:

http://www.open-mpi.org/faq/?category=running.

Run "shmemrun --help" to obtain ScalableSHMEM job launcher runtime parameters.

ScalableSHMEM contains support for environment module system (http://modules.sf.net/). The modules configuration file can be found at: /opt/mellanox/openshmem/2.2/etc/shmem_modulefile

2.3.1 Basic ScalableSHMEM Job Run Example

- ***To launch ScalableSHMEM application, run:***

```
% /opt/mellanox/openshmem/2.2/bin/shmemrun -np 2 --bind-to-core -bynode -display-map -
hostfile myhostfile /opt/mellanox/openshmem/2.2/bin/shmem_osu_latency
```

The example above shows how to run 2 copies of shmem_osu_latency program on hosts specified in the 'myhostfile' file.

3 Running ScalableSHMEM

3.1 Running ScalableSHMEM with MXM

MellanoX Messaging (MXM) library provides enhancements to parallel communication libraries by fully utilizing the underlying networking infrastructure provided by Mellanox HCA/switch hardware. This includes a variety of enhancements that take advantage of Mellanox networking hardware including:

- Multiple transport support including RC, XRC and UD
- Proper management of HCA resources and memory structures
- Efficient memory registration
- One-sided communication semantics
- Connection management
- Receive side tag matching
- Intra-node shared memory communication

These enhancements significantly increase the scalability and performance of message communi-cations in the network, alleviating bottlenecks within the parallel communication libraries

3.1.1 Enabling MXM for ScalableSHMEM Jobs

MXM is activated automatically in ScalabeSHMEM for jobs with Number of Elements (PE) higher or equal to 128.

➢ ***To enable MXM for SHMEM jobs for any PE:***

- Add the following MCA parameter to the `shmemrun` command line.

```
-mca spml_ikrit_np <number>
```

➢ ***To force MXM usage:***

- Add the following MCA parameter `shmemrun` command line.

```
-mca spml_ikrit_np 0
```

For additional MXM tuning information, please refer to the MellanoX Messaging Library README file found in the Mellanox website.

3.1.2 Working with Multiple HCAs

If there several HCAs in the system, MXM will choose the first HCA with the active port to work with. The HCA/port to be used can be specified by setting the `MXM_RDMA_PORTS` environment variable. The variable format is as follow: `MXM_RDMA_PORTS=hca_name:port,...`

For example, the following will cause MXM to use port one on two installed HCAs:

```
MXM_RDMA_PORTS=mlx4_0:1,mlx4_1:1
```

The environment variables must be run via the shmemrun command line:

```
shmemrun -x MXM_RDMA_PORTS=mlx4_0:1 ...
```

3.2 Running ScalableSHMEM with RC Transport

In general RC QP gives a better performance when number of nodes and number of ranks per node are small. RC transport is used by default if the number of ranks is no greater than 128.

➢ ***To turn off MXM and use connection oriented QP transport:***

```
-mca spml yoda
```

3.2.1 Working with Multiple HCAs

When SHMEM is in RC mode, the first active port will always be used for data traffic.

The HCA/port to be used can be specified by setting the `-mca btl_openib_if_include hca_name:port` in the `shmemrun`.

3.3 Running ScalableSHMEM with FCA

The Mellanox Fabric Collective Accelerator (FCA) is a unique solution for offloading collective operations from the Message Passing Interface (MPI) or ScalableSHMEM process onto Mellanox InfiniBand managed switch CPUs. As a system-wide solution, FCA utilizes intelligence on Mellanox InfiniBand switches, Unified Fabric Manager and MPI nodes without requiring additional hardware. The FCA manager creates a topology based collective tree, and orchestrates an efficient collective operation using the switch-based CPUs on the MPI/ScalableSHMEM nodes.

FCA accelerates MPI/ScalableSHMEM collective operation performance by up to 100 times providing a reduction in the overall job runtime. Implementation is simple and transparent during the job runtime.

FCA is disabled by default and *must* be configured prior to using it from the ScalableSHMEM.

➢ ***To enable FCA by default in the ScalableSHMEM:***

1. Edit the `/opt/mellanox/openshmem/2.2/etc/openmpi-mca-params.conf` file.
2. Set the `scoll_fca_enable` parameter to `1`.

```
scoll_fca_enable=1
```

3. Set the `scoll_fca_np` parameter to 0.

```
scoll_fca_np=0
```

➢ ***To enable FCA in the*** `shmemrun` ***command line, add the following:***

```
-mca scoll_fca_enable=1
-mca scoll_fca_enable_np 0
```

➢ ***To disable FCA:***

```
-mca scoll_fca_enable 0 -mca coll_fca_enable 0
```

For more details on FCA installation and configuration, please refer to the FCA User Manual found in the Mellanox website.

3.4 Developing Application using ScalableSHMEM together with MPI

The SHMEM programming model can provide a means to improve the performance of latency-sensitive sections of an application. Commonly, this requires replacing MPI send/recv calls with `shmem_put/ shmem_get` and `shmem_barrier` calls. The SHMEM programming model can deliver significantly lower latencies for short messages than traditional MPI calls. An alternative to `shmem_get /shmem_put` calls can also be considered the MPI-2 MPI_Put/ MPI_Get functions.

An example of MPI-SHMEM mixed code.

```
/* example.c */

#include <stdlib.h>
#include <stdio.h>
#include "shmem.h"
#include "mpi.h"
int main(int argc, char *argv[])
{
 MPI_Init(&argc, &argv);
    start_pes(0);

 {
        int version = 0;
        int subversion = 0;
        int num_proc = 0;
        int my_proc = 0;
        int comm_size = 0;
        int comm_rank = 0;

        MPI_Get_version(&version, &subversion);
        fprintf(stdout, "MPI version: %d.%d\n", version, subversion);

        num_proc = _num_pes();
        my_proc = _my_pe();

        fprintf(stdout, "PE#%d of %d\n", my_proc, num_proc);

        MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
        MPI_Comm_rank(MPI_COMM_WORLD, &comm_rank);

        fprintf(stdout, "Comm rank#%d of %d\n", comm_rank, comm_size);
    }

    return 0;
}
```

3.5 ScalableSHMEM Tunable Parameters

ScalableSHMEM uses Modular Component Architecture (MCA) parameters to provide a way to tune your runtime environment. Each parameter corresponds to a specific function. The following are parameters that you can change their values to change the application's the function:

- memheap - controls memory allocation policy and thresholds
- scoll - controls ScalableSHMEM collective API threshold and algorithms

- spml - controls ScalableSHMEM point-to-point transport logic and thresholds
- atomic - controls ScalableSHMEM atomic operations logic and thresholds
- shmem - controls general ScalableSHMEM API behavior

➢ ***To display ScalableSHMEM parameters:***

1. Print all available parameters. Run:

```
/opt/mellanox/openshmem/2.2/bin/shmem_info -a
```

2. Print ScalableSHMEM specific parameters. Run:

```
/opt/mellanox/openshmem/2.2/bin/shmem_info --param shmem all
/opt/mellanox/openshmem/2.2/bin/shmem_info --param memheap all
/opt/mellanox/openshmem/2.2/bin/shmem_info --param scoll all
/opt/mellanox/openshmem/2.2/bin/shmem_info --param spml all
/opt/mellanox/openshmem/2.2/bin/shmem_info --param atomic all
```

3.5.1 OpenSHMEM MCA Parameters for Symmetric Heap Allocation

SHMEM memheap size can be modified by adding the `SHMEM_SYMMETRIC_HEAP_SIZE` parameter to the `shmemrun` file. The default heap size is 256M.

➢ ***To run SHMEM with memheap size of 64M. Run:***

```
/opt/mellanox/openshmem/2.2/bin/shmemrun -x SHMEM_SYMMETRIC_HEAP_SIZE=64M -np 512 -
mca mpi_paffinity_alone 1 -bynode -display-map -hostfile myhostfile example.exe
```

➢ ***To allocate symmetric heap with huge pages. Run:***

```
shmemrun -np 512 -mca shmalloc_use_hugepages 5 -bynode -hostfile myhostfile exam-
ple.exe
```

`shmalloc_use_hugepages` values are: 0, 1, 2, 5, 100, and 101 (default is 1).

- 0 - Allocates symmetric heap with sysv shmget()
- 1 - Allocates symmetric heap using the following allocators:
  - ib verbs continuous memory
  - hugepages by giving SHM_HUGETLB to shmget()
  - regular shmget()
- 2 - Allocates symmetric heap with huge pages.
- 5 - Uses ib verbs contiguous memory API

If older kernel versions are used (<2.6.32) that do not allow performing the shmat() action on the deleted segments, the following shared memory values are used:

- 3 - (same as value 2) but does NOT immediately remove sysv segment id with shmctl(IPC_RMID).
- 4 - (same as value 0) but does NOT immediately remove sysv segment id with shmctl(IPC_RMID).

3.5.2 Parameters Used to Force Connection Creation

Commonly SHMEM creates connection between PE lazily. That is at the sign of the first traffic.

➢ ***To force connection creating during startup:***

- Set the following MCA parameter.

```
-mca shmem_preconnect_all 1
```

Memory registration (ex: infiniband rkeys) information is exchanged between ranks during startup.

➢ ***To enable on demand memory key exchange:***

- Set the following MCA parameter.

```
-mca shmalloc_use_modex 0
```

3.6 Configuring ScalableSHMEM with External Profiling Tools (TAU)

ScalableSHMEM supports external Tuning and Profiling tool TAU.

For further information, please refer to http://www.cs.uoregon.edu/Research/tau/home.php

3.6.1 Using TAU with OpenSHMEM

3.6.1.1 Building a PDT Toolkit

1. Download the PDT Toolkit.

```
wget -nc http://tau.uoregon.edu/pdt_releases/pdtoolkit-3.17.tar.gz
tar -xzf pdtoolkit-3.17.tar.gz
```

2. Configure and build the PDT toolkit.

```
cd pdtoolkit-3.17
PDT_INST=$PWD
./configure --prefix=/usr/local
make install
```

3.6.1.2 Building a TAU Toolkit

1. Download the TAU Toolkit..

```
wget -nc http://www.cs.uoregon.edu/research/paracomp/tau/tauprofile/dist/
tau_latest.tar.gz
tar -xzf tau_latest.tar.gz
```

2. Configure and build the TAU toolkit.

```
cd tau-latest
TAU_SRC=$PWD
patch -p1 -i /opt/mellanox/openshmem/2.2/share/openshmem/tau_openshmem.patch
OSHMEM_INST=/opt/mellanox/openshmem/2.2
TAU_INST=$TAU_SRC/install
./configure -prefix=$TAU_INST -shmem -tag=oshmem -cc=gcc -pdt=$PDT_INST -PROFILEPARAM
-useropt="-I$OSHMEM_INST/include/mpp" -shmemlib=$OSHMEM_INST/lib -shmemlibrary="-lshmem
-lpmi"
make install
```

The patch is required to define a profiling API that is not part of an official openshmem.org standard.

Appendix A: Performance Optimization

A.1 Configuring Hugepages

Hugepages is a feature applicable to users using MLNX_OFED v1.5.3-3.0.0.

Hugepages can be allocated using the `/proc/sys/vm/nr_hugepages` entry, or by using the sysctl command.

- ***To view the current setting using the /proc entry:***

```
cat /proc/sys/vm/nr_hugepages
0
```

- ***To view the current setting using the sysctl command:***

```
sysctl vm.nr_hugepages
vm.nr_hugepages = 0
```

- ***To set the number of huge pages using /proc entry:***

```
echo 1024 > /proc/sys/vm/nr_hugepages
```

- ***To set the number of hugepages using sysctl :man***

```
sysctl -w vm.nr_hugepages=1024
vm.nr_hugepages = 1024
```

To allocate all the hugepages needed, you might need to reboot your system since the hugepages requires large areas of contiguous physical memory.

In time, physical memory may be mapped and allocated to pages, thus the physical memory can become fragmented. If the hugepages are allocated early in the boot process, fragmentation is unlikely to have occurred.

It is recommended that the `/etc/sysctl.conf` file be used to allocate hugepages at boot time.

For example, to allocate 1024 hugepages at boot time, add the line below to the sysctl.conf file:

```
vm.nr_hugepages = 1024
```

A.2 Tuning MTU Size to the Recommended Value

The procedures described below apply to user using MLNX_OFED 1.5.3.-3.0.0 only.

When using MLNX_OFED 1.5.3-3.0.0, it is recommended to change the MTU to 4k. Whereas in MLNX_OFED 1.8 the MTU is already set by default to 4k.

➢ ***To check the current MTU support of an InfiniBand port, use the smpquery tool:***

```
# smpquery -D PortInfo 0 1 | grep -i mtu
```

If the MtuCap value is lower than 4K, enable it to 4K.

Assuming the firmware is configured to support 4K MTU, the actual MTU capability is further limited by the mlx4 driver parameter.

➢ ***To further tune it:***

1. Set the `set_4k_mtu` mlx4 driver parameter to 1 on all the cluster machines. For instance:

```
# echo "options mlx4_core set_4k_mtu=1" >> /etc/modprobe.d/mofed.conf
```

2. Restart openibd.

```
# service openibd restart
```

➢ ***To check whether the parameter was accepted, run:***

```
# cat /sys/module/mlx4_core/parameters/set_4k_mtu
```

To check whether the port was brought up with 4K MTU this time, use the smpquery tool again.

A.3 HPC Applications on Intel Sandy Bridge Machines

Intel Sandy Bridge machines have NUMA hardware related limitation which affects performance of HPC jobs utilizing all node sockets. When installing MLNX_OFED 1.8, an automatic work-around is activated upon Sandy Bridge machine detection, and the following message is printed in the job`s standard output device: `"mlx4: Sandy Bridge CPU was detected"`

➢ ***To disable MOFED 1.8 Sandy Bridge NUMA related workaround:***

- Set the SHELL environment variable before launching HPC application. Run:

```
% export MLX4_STALL_CQ_POLL=0
% shmemrun <...>
```

or

```
shmemrun -x MLX4_STALL_CQ_POLL=0 <other params>
```